Security Audit

Drop Security Audit (originally FontelePay)

> **Rebrand note:** FontelePay was renamed to Drop. This audit predates the backend implementation and PSD2 pass-through architecture. Many issues identified here (localStorage PIN, client-side auth) have been addressed in the current backend. See `docs/security/SECURITY-ARCHITECTURE.md` for current security posture.

Date: 2026-02-05 **Auditor:** John (AI Director) **Scope:** Full MVP application scan (pre-backend, client-side only)

Executive Summary

Category	Status	Issues
Dependencies	PASS	0 vulnerabilities
Hardcoded Secrets	PASS	None found
XSS Prevention	PASS	No dangerous patterns
Data Storage	⚠ WARNING	PIN in localStorage
Authentication	⚠ WARNING	Client-side only
Input Validation	⚠ WARNING	Minimal validation

Overall: ACCEPTABLE FOR MVP DEMO, NOT PRODUCTION READY

---

1. Dependency Audit

```
$ npm audit
found 0 vulnerabilities
```

 All npm packages are up to date with no known vulnerabilities.

---

2. Secret Management

Checked For:

- Hardcoded API keys
- Hardcoded passwords
- Hardcoded tokens
- .env files in repo

Findings:

 No hardcoded secrets in source code No .env files committed Mock tokens are clearly labeled as mocks

Recommendation:

- Create `.env.example` template for production
- Use environment variables for real API keys
- Add `.env*` to .gitignore (already present)

---

3. XSS Prevention

Checked For:

- `dangerouslySetInnerHTML`
- `eval()`
- Direct `innerHTML` manipulation

Findings:

 No dangerous DOM manipulation patterns found React's built-in XSS protection active

4. Data Storage (CRITICAL)

Issue: PIN Stored in localStorage

```
// AppContext.tsx:87
localStorage.setItem(STORAGE_KEY, JSON.stringify(user));
// user object includes plaintext PIN!
```

Risk Level: **HIGH** for production

Why It Matters:

- localStorage accessible via XSS attacks
- localStorage persists after browser close
- PIN visible in browser DevTools
- No encryption

Fix Required for Production:

```
// Option 1: Never store PIN client-side
// Use server-side session + httpOnly cookies

// Option 2: Store hashed PIN (still not ideal)
import bcrypt from 'bcryptjs';
const hashedPin = await bcrypt.hash(pin, 10);

// Option 3: Use Web Crypto API
const encoder = new TextEncoder();
const data = encoder.encode(pin);
const hash = await crypto.subtle.digest('SHA-256', data);
```

Immediate Mitigation:

- For MVP demo: ACCEPTABLE (no real money)
- For production: MUST implement server-side auth

5. Authentication Analysis

Current State:

- □ PIN required for login
- □ Wrong PIN rejected
- ⚠ No rate limiting on PIN attempts
- ⚠ No session expiry
- ⚠ No logout on inactivity
- □ No server-side verification

OWASP Top 10 Review:

Vulnerability	Status	Notes
A01 Broken Access Control	⚠	No server-side checks
A02 Cryptographic Failures	□	PIN not hashed
A03 Injection	□	React prevents XSS
A04 Insecure Design	⚠	Client-only auth
A05 Security Misconfiguration	□	N/A for static
A06 Vulnerable Components	□	npm audit clean
A07 Auth Failures	⚠	No rate limiting
A08 Data Integrity	□	No deserialization
A09 Logging Failures	⚠	No audit logging
A10 SSRF	□	No server requests

---

6. Input Validation

Checked Areas:

Input	Validation	Status
Phone number	Length >= 6	⚠ Weak
OTP	Length === 6	⚠ No server verify

Input	Validation	Status
PIN	Length === 4	✅ OK
Transfer amount	> 0, <= balance	✅ OK
IBAN	None	⚠ Should validate format
Card number	Length === 16	⚠ No Luhn check

Recommendations:

- Add IBAN format validation (checksum)
- Add Luhn algorithm for card numbers
- Add phone number format validation
- Implement server-side validation

---

7. Production Readiness Checklist

Must Have Before Launch:

- ☐ Server-side authentication (NextAuth.js or similar)
- ☐ Hashed/encrypted PIN storage
- ☐ HTTPS enforcement
- ☐ Rate limiting on auth endpoints
- ☐ Session management with expiry
- ☐ Audit logging
- ☐ Error monitoring (Sentry)
- ☐ CSP headers
- ☐ CORS configuration

Nice to Have:

- ☐ 2FA / MFA
- ☐ Device fingerprinting
- ☐ Anomaly detection
- ☐ PCI-DSS compliance audit

---

8. Recommendations Summary

Immediate (Before any real users):

1. Move authentication to server-side
2. Hash PINs with bcrypt
3. Add rate limiting

Short-term (Before production):

4. Implement proper session management
5. Add audit logging
6. Setup error monitoring
7. Add input validation

Long-term (For compliance):

8. PCI-DSS audit for card handling
9. Penetration testing
10. Security certification

Conclusion

The Drop MVP (originally FontelePay) is **acceptable for demonstration purposes** but requires significant security improvements before handling real user data or money.

Key Risk: Plaintext PIN storage in localStorage

Mitigation: This is a mock/demo environment with no real financial transactions. All "money" is simulated.

Next Steps: Implement server-side auth before any production deployment.

Audit completed by automated security scan + manual code review.

Revision #5
Created 2026-02-18 08:44:31 UTC by John
Updated 2026-05-25 07:23:49 UTC by John